(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
3 January 2002 (03.01.2002)

PCT

(10) International Publication Number
WO 02/01356 A1

(51) International Patent Classification⁷: G06F 9/54

(21) International Application Number: PCT/US01/19007

(22) International Filing Date: 12 June 2001 (12.06.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/599,510 23 June 2000 (23.06.2000) US

(71) Applicant: PREVIEW SYSTEMS,INC. [US/US]; 1195 W. Fremont Avenue, Suite 2001, Sunnyvale, CA 94087 (US).

(72) Inventors: TAYLOR, Christopher, S.; 10317 Mary Avenue, Cupertino, CA 95014 (US). KIMMET, Timothy, Gordon; 15660 El Gato Lane, Los Gatos, CA 95032 (US).

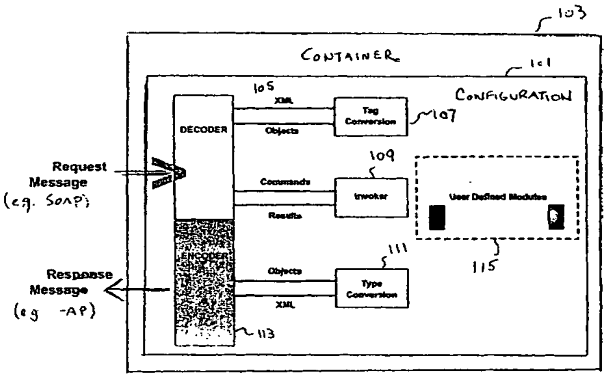
(74) Agent: KREBS, Robert, E.; Burns, Doane, Swecker & Mathis, LLP, P.O. Box 1404, Alexandria, VA 22313-1404 (US).

(81) Designated State *(national)*: JP.

(84) Designated States *(regional)*: European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR).

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

(54) Title: TYPE CONVERSION TECHNIQUE FACILITATING REMOTE SERVICE INVOCATION



(57) Abstract: The present invention, generally speaking, provides a messaging framework (101) including a flexible type conversion facility (107, 111) for use in remote method invocation via the Internet. When the framework (101) receives a message, a request message decoder (105) decodes the message; a tag conversion routine (107) converts any parameters into objects, locates the appropriate message handler; and an invoker module (107) invokes the requested method on that handler, and communicates a return value (or exception) via a response message. In the case of the SOAP protocol, for example, the return value (or exception) is encoded into XML and wrapped in a SOAP protocol response message. Dynamic type conversion (111) of parameters into objects is performed using a tag library system (107). In the tag library system (107), special user-defined modules (115) (called "type factories") are handed XML elements for conversion into objects. Unlike a strictly type-conversion architecture such as classic object serialization, there are no dependencies between the XML input and the objects produced by the type factory. Furthermore, the factory is not required to extract all of the content during a conversion, but can pass the raw XML content to the constructed object for later use.

WO 02/01356 A1

TYPE CONVERSION TECHNIQUE FACILITATING REMOTE SERVICE INVOCATION

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to distributed computing, in particular client/server communication and server/server communication.

2. State of the Art

Various techniques enabling software in one location to invoke the services of software at another location are known, including, for example, RPCs (Remote Procedure Calls), Java RMI (Remote Method Invocation), CORBA (Common Object Request Broker Architecture), and Microsoft COM (Common Object Model). Many of these techniques entail substantial complexity. Furthermore, in the case of COM, for example, COM components can only run on Microsoft Operating Systems, and COM requests cannot pass through a firewall. In contrast to the complexity of the foregoing solutions, XML (eXtensible Markup Language) has surged in popularity, due at least in part to its simplicity and flexibility.

The SOAP (Simple Object Access Protocol) protocol is XML-based and enables a client's request to be communicated over HTTP, across the Internet and through firewalls. XML documents are used for encoding of the invocation requests and responses. In the SOAP protocol, a method request is represented as an XML element with additional elements for parameters. The SOAP protocol considerably simplifies remote method invocation. Nevertheless, SOAP makes no provision for type conversion, i.e., mapping between XML and data types usable by the method being invoked and, in the reverse direction, between data types and XML. Two techniques are known. One technique encodes data types through "introspection," i.e., programmatically inspecting an object, walking through its member variables, and turning each into XML. In another technique, no mapping between data types and XML is performed at all. Rather, raw XML is simply passed to the method. Doing so makes for messy code within the method and has a tendency to defeat the discipline of good object-oriented coding practices. Hence, these techniques are fairly cumbersome.

Also, in these prior art mechanisms, versioning issues arise in which, if changes in services are made on the server side, corresponding changes must be made to each individual client to maintain operability. Programmers making changes to services must therefore concern themselves with client/server business logic, potentially a substantial burden.

Therefore, there remains a need for a flexible type conversion facility for use in remote method invocation, e.g., in conjunction with the SOAP protocol.

SUMMARY OF THE INVENTION

The present invention, generally speaking, provides a messaging framework including a flexible type conversion facility for use in remote method invocation via the Internet. When the framework receives a message, it decodes the message, converts any parameters into objects, locates the appropriate message handler, invokes the requested method on that handler, and communicates a return value (or exception) via a response message. In the case of the SOAP protocol, for example, the return value (or exception) is encoded into XML and wrapped in a SOAP protocol response message. Dynamic type conversion of parameters into objects is performed using a tag library system. In the tag library system, special user-defined modules (called "type factories") are handed XML elements for conversion into objects. Unlike a strict type-conversion architecture such as classic object serialization, there are no dependencies between the XML input and the objects produced by the type factory. Furthermore, the factory is not required to extract all of the content during a conversion, but can pass the raw XML content to the constructed object for later use.

BRIEF DESCRIPTION OF THE DRAWING

The present invention may be further understood from the following description in conjunction with the appended drawing. In the drawing:

Figure 1 is a block diagram of one computing environment using the present messaging framework;

Figure 2 is a diagram showing the organization of a tag library used to perform conversion between XML and code objects;

Figure 3 is a conceptual diagram illustrating conversion from XML;

Figure 4 is a block diagram of another computing environment using the present messaging framework; and

Figure 5 is a block diagram showing in greater detail various components of the present messaging framework.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring now to Figure 1, a block diagram is shown of one computing environment using the present messaging framework, referred to as the "Ivory" messaging framework. In Figure 1, the messaging framework (configuration 101) is shown as residing within a container 103. An example of one such container is a "servlet," i.e., an extension similar to a Java applet except that it runs on the server instead of the client. Servlets are described in greater detail in Taylor and Kimmet, *Core Java Web Server*, Prentice Hall, 1999, incorporated herein by reference.

The messaging framework consists of a collection of components which, when initialized to an operational state, form a configuration of the messaging framework. These components include, as shown in Figure 1, a request message decoder 105, tag conversion routines 107, an invoker module 109, type conversion routines 111, and a response message encoder 113. The messaging framework interacts with various user-define modules 115 that perform the desired services.

The message request decoder 105 receives an XML request message, e.g., a SOAP message, decodes the message, and for different tags of the XML message, passes the XML for the tag to a tag conversion routine 107. Tag conversion routines 107 receive as input the XML for a tag and produce as output corresponding code objects (e.g., Java objects). The message request decoder 105 makes these objects available to the invoker module 109 along with a corresponding command. The invoker module 109 receives the command and invokes a corresponding user defined module 115 to perform the actual requested work. The invoker modules 109 receives command results, in the form of objects, from the user defined module 115 and returns these results to the response message encoder 105.

The response message encoder 113 receives the result objects and for different ones of the objects, passes the object to a corresponding type conversion routine 111. Type conversion routines 111 perform the opposite function of tag conversion routines, i.e., receive as input an object and produce as output corresponding XML.

Finally, the response message encoder 113 uses the XML to create an XML message, e.g, a SOAP message.

In general, when a SOAP message is handed to the messaging framework, the framework performs the following steps:

1. Decode the message.
2. Convert any parameters to objects.
3. Locate the appropriate handler.
4. Invoke the method on that handler.
5. Encode the return value (or exception) into XML.

6. Wrap the return value in a SOAP response message.

Whereas SOAP is only a protocol-layer construction, the present messaging framework provides for the separation and distinction of different aspects of remote method invocation, potentially allowing for these different aspects to be addressed by different programmers or developers. For example, an invoker developer would be concerned with what gets done in a request, a tag conversion developer would be concerned with what kinds of data can be passed in a request, and a protocol handler developer would be concerned with how messages get into and out of the messaging framework.

An important feature of the messaging framework is a "Tag Library" system in which special user defined modules (referred to sometimes hereafter as "type factories") are handed XML elements for conversion into objects. Unlike a strict type-conversion architecture like classical object serialization, there are no dependencies between what XML enters the framework and the objects produced by each user defined module. Furthermore, the user defined modules are not required to extract all of the content during a conversion, but can pass raw XML content to the constructed object for later use.

To use the messaging framework, a configuration is first built. In an exemplary embodiment, the configuration process begins by reading an ivory.config file, the content of which is specified in XML format. The name of an XML top-level tag and its associated tag (*toXML*) converter are specified, as well as the name of a Java object and its type (*fromXML*) converter. In the example of Figure 2, two converters are specified: string, which is a primitive type converter for a Java string, and Product, which is a user-defined type for a more complex object.

The messaging framework loads the ivory.config file and creates a tag library (converter library) having two types of converters: *toXML* converters, which map an object type to its corresponding converter, or "type factory," and *fromXML* converters, which map an XML tag and associated elements to its corresponding converter.

In operation, the messaging framework converts objects from XML by matching the top-level XML tag with its associated converter. Referring to Figure 3, an example of shown of how, using the tag library of Figure 2, a top-level Product element is converted from XML into a PreviewProduct using the PreviewProductType converter. The messaging framework performs the following tasks in order to convert the XML representation of the Product into a PreviewProduct Java object:

1. Get the name of the top-level tag, which is "Product."
2. Use the fromXML portion of the tag library to look up the converter for the XML tag with the name "Product." This step renders an instance of the PreviewProductType converter.
3. Invoke fromXML on the converter, which returns a Java object that is an instance of PreviewProduct.

Note that a complex type might contain simple types. In this instance, a converter for the complex type might use a converter for the simple type to help perform the conversion. Note also that converter need not have knowledge of every element to perform a conversion; unknown elements may simply be passed along unconverted to a user defined module for the user defined module to handle. Considerable flexibility therefore results.

Appendix A shows a type converter that extracts text content and returns a String object.

The Ivory framework is not container dependent. Hence, although the HTTP represents a convenient container with which the messaging framework may be used, any other desired system may be used. For example, Figure 4 shows the messaging framework used with email, in accordance with any of various email protocols.

Referring to Figure 5, a block diagram shows in greater detail various components of the present messaging framework. The encoder/decoder 501 performs the functions of the encoder and decoder in the representation of Figure 1. Where the messaging framework is used with the SOAP protocol, for example, the encoder/decoder may represent the SOAP protocol encoding/decoding layer. SOAP messages, as remote method invocations, contain typed data. These data types are converted into objects before leaving the encoding/decoding layer 501.

The responsibility for converting data types into objects falls on the type library 503. The type library 503 performs the functions of tag conversion and type conversion in Figure 1. The type library maps incoming XML data elements to type factories 505. When an XML element arrives that has an associated type factory, that factory builds a Java object from the XML. The type factories are the converter routines of Figure 2.

The purpose of the messaging framework is to make remote method invocation easy to accomplish. In the case of the present messaging framework, remote method invocation is used to invoke the services of request handlers. A handler request is a remote object that exposes specific methods to the messaging framework. Maintaining these objects and routing incoming method calls is the responsibility of the handler registry 509. The handler registry performs the invoker functions of Figure 1. The handler objects correspond to the user defined modules of Figure 1. In an exemplary embodiment, the messaging framework does not require its handler objects to implement a fixed interface. Rather, all a handler object needs to expose a method to the messaging framework is to prefix the word "soap" to the front of any public method. During initialization of the messaging framework, handler objects are loaded and bound into an invoker object. During the binding process, the handler object undergoes introspection, in which all of the public methods of the handler object are inspected by name. Any methods starting with the keyword "soap" are exported for remote invocation purposes as illustrated in the code example of Appendix B.

The configuration services block 511 receives input from a configuration file as described in connection with Figure 2. Rather than using a simple text file for configuring the messaging framework, however, all framework objects are extracted from a configurator. In an exemplary embodiment, the configurator parses a DOM tree for configuration information.

In addition, there is shown in Figure 5 an optional logging services block 515 that produces a log file 517, and an optional database connection manager 519. The database connection manager allows handler objects to make database connections. In the example of Figure 5, the database connection manager connects to a Ziplock database 521 associated with a ZiplockTM electronic software distribution server of the present assignee.

A code example illustrating invocation of the desired method is provided as Appendix C.

It will be appreciated by those of ordinary skill in the art that the invention can be embodied in other specific forms without departing from the spirit or essential character thereof. The presently disclosed embodiments are therefore considered in all respects to be illustrative and not restrictive. The scope of the invention is indicated by the appended claims rather than the foregoing description, and all changes which come within the meaning and range of equivalents thereof are intended to be embraced therein.

What is claimed is:

1. A method whereby one computer remotely invokes a service by another computer, wherein a message is used to construct an object to be used by the other computer in providing the service, the method comprising the steps of:

providing, accessible to the other computer, a tag library comprising, for each of multiple tag values, a named routine for constructing respective corresponding objects;

the one computer producing a message including tags denoting types, including at least one tag denoting one type to be used to construct an object, and, corresponding to each tag, text representing one or more element values, the message including an identifier of a requested service;

decoding the message;

for said one tag, calling a corresponding routine from the tag library, the routine constructing an object using one or more elements values of said one type; and

making the object available to a handler for performing the requested service, and calling the handler.

2. The method of Claim 1, wherein the tag library comprises, for each of multiple data types, a named routine for producing a tag/elements representation of the data type, the method further comprising:

the handler performing the requested service and returning result or status information including at least one data type;

for said one data type, call a corresponding routine from the tag library, the routine producing a representation of the data type as a tag and, corresponding to the tag, text representing one or more element values; and

sending a response message including the tag and element values.

3. The method of Claim 1, wherein the message is a SOAP message.

4. The method of Claim 2, wherein the response message is a SOAP message.

5. The method of Claim 1, wherein the message is an email message.

6. The method of Claim 2, wherein the response message is an email message.

7. The method of Claim 1, wherein, in addition to making the object available to a handler, other tag and element values not used to create the object are made available to the handler.

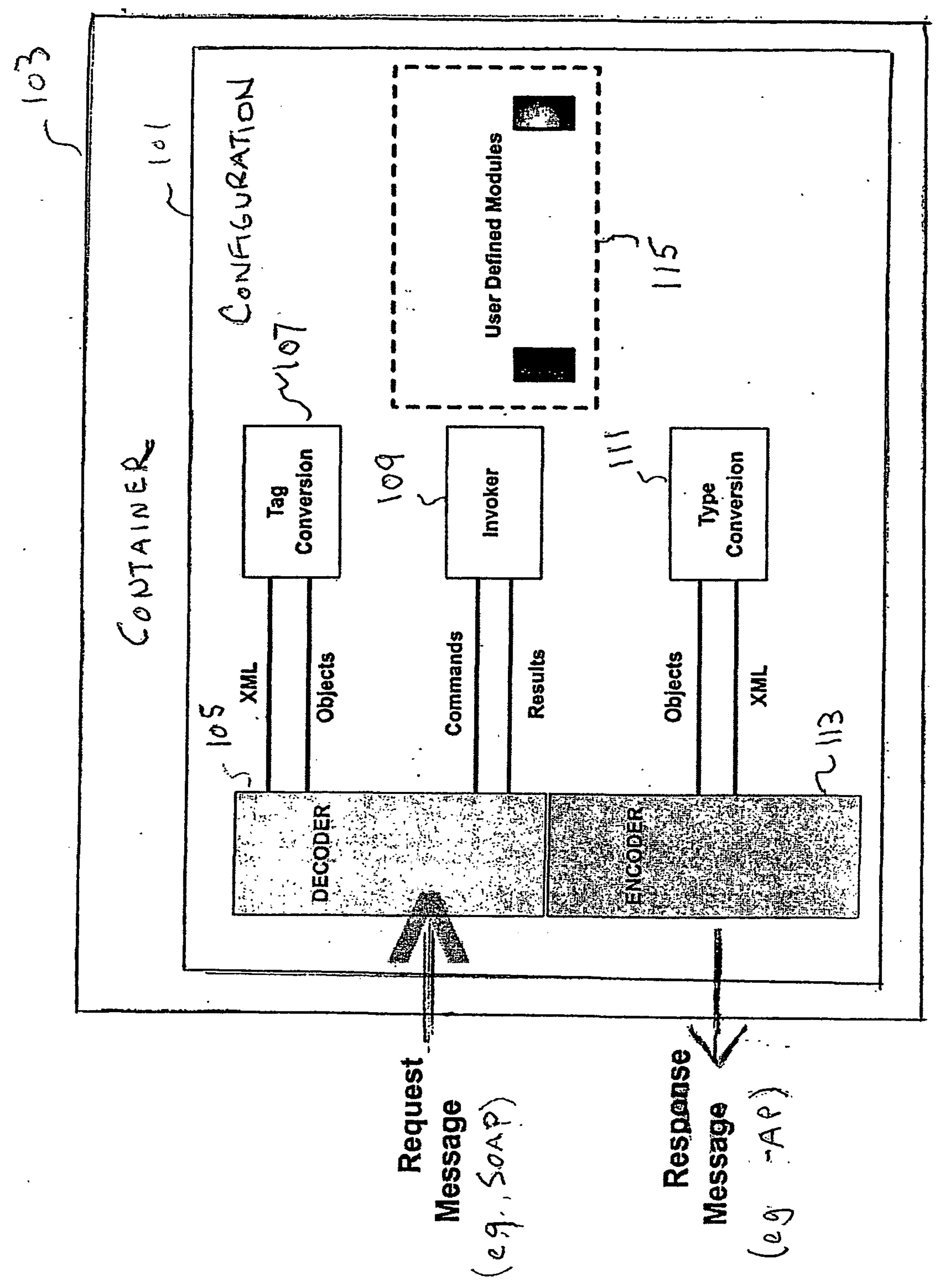
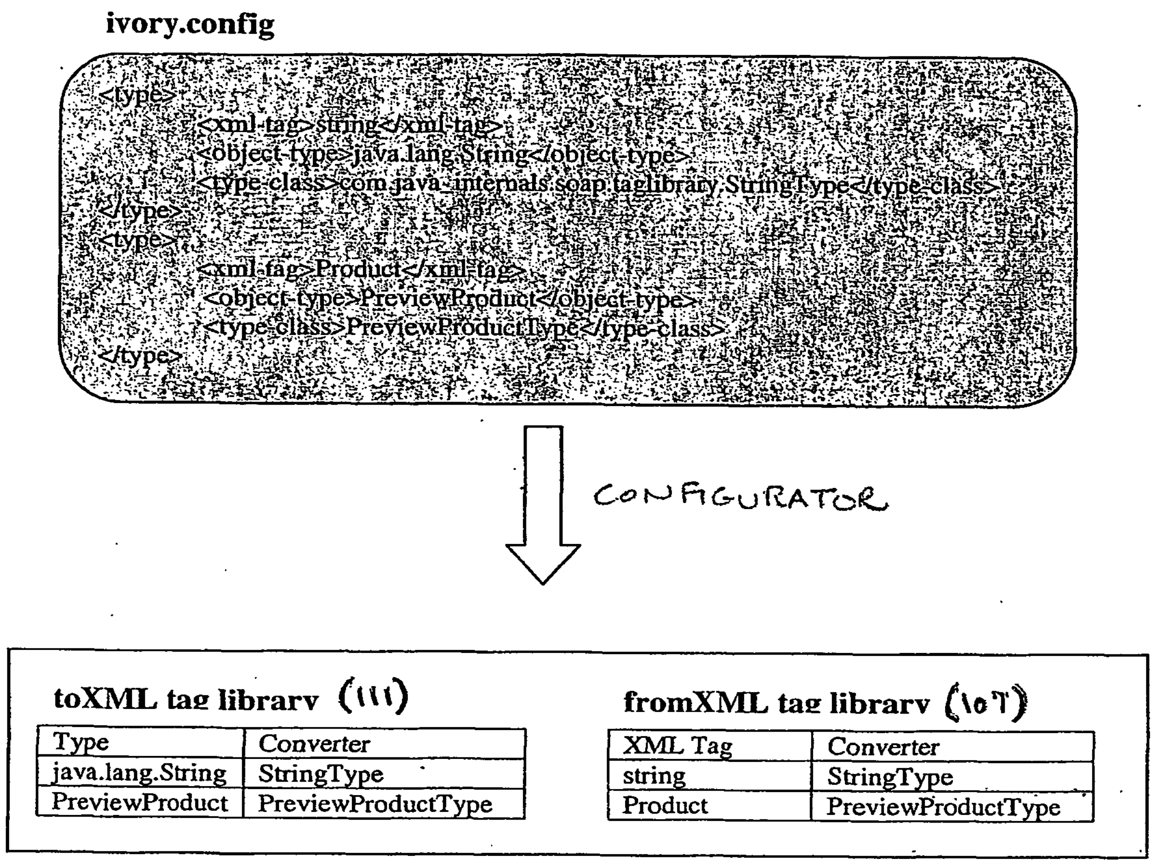

103
CONTAINER
101
CONFIGURATION
107
Tag Conversion
109
Invoker
111
Type Conversion
User Defined Modules
115
105
XML
Objects
Commands
Results
Objects
XML
113
DECODER
ENCODER
Request Message (eg, SOAP)
Response Message (eg -AP)

Fig 1



ivory.config

```
<type>
        <xml-tag>string</xml-tag>
        <object-type>java.lang.String</object-type>
        <type-class>com.java_internals.soap.taglibrary.StringType</type-class>
</type>
<type>
        <xml-tag>Product</xml-tag>
        <object-type>PreviewProduct</object-type>
        <type-class>PreviewProductType</type-class>
</type>
```

toXML tag librarv (111)

Type	Converter
java.lang.String	StringType
PreviewProduct	PreviewProductType

fromXML tag librarv (107)

XML Tag	Converter
string	StringType
Product	PreviewProductType

FIG. 2

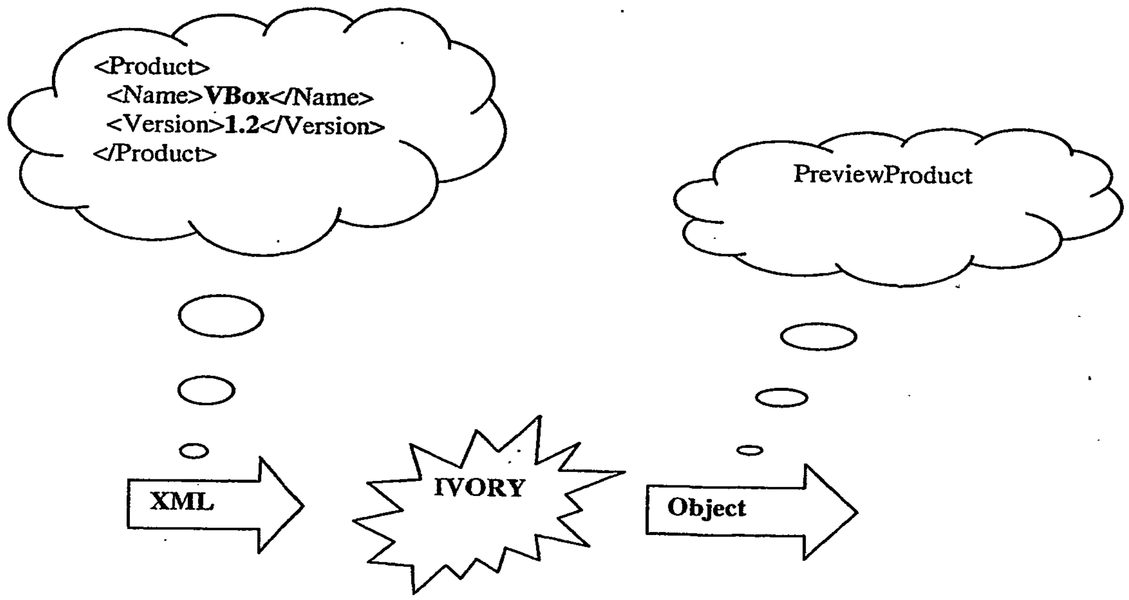


FIG. 3

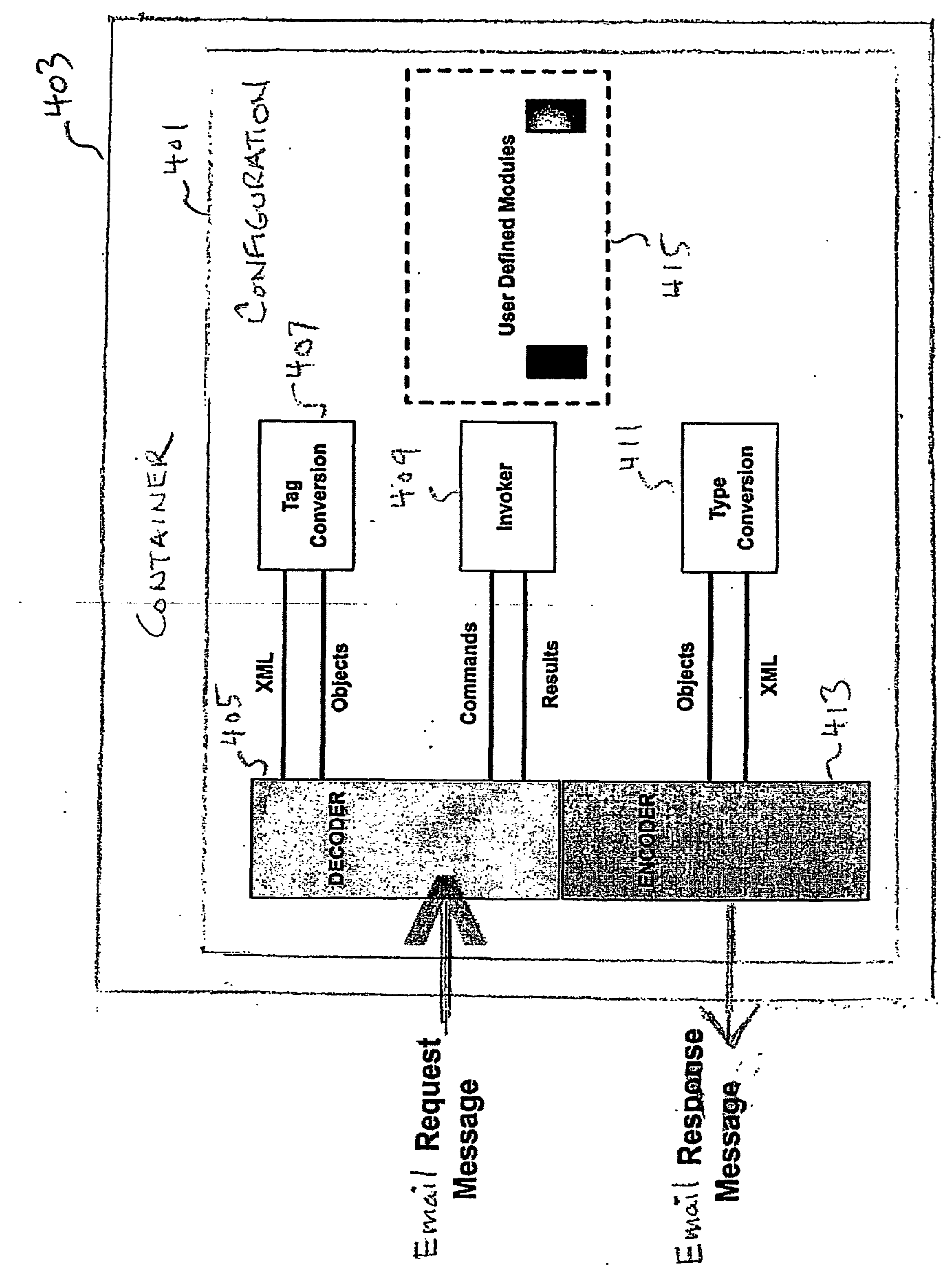

403
401
CONFIGURATION
CONTAINER
User Defined Modules
415
407
Tag Conversion
409
Invoker
411
Type Conversion
XML
Objects
Commands
Results
Objects
XML
405
413
DECODER
ENCODER
Email Request Message
Email Response Message

Fig 4

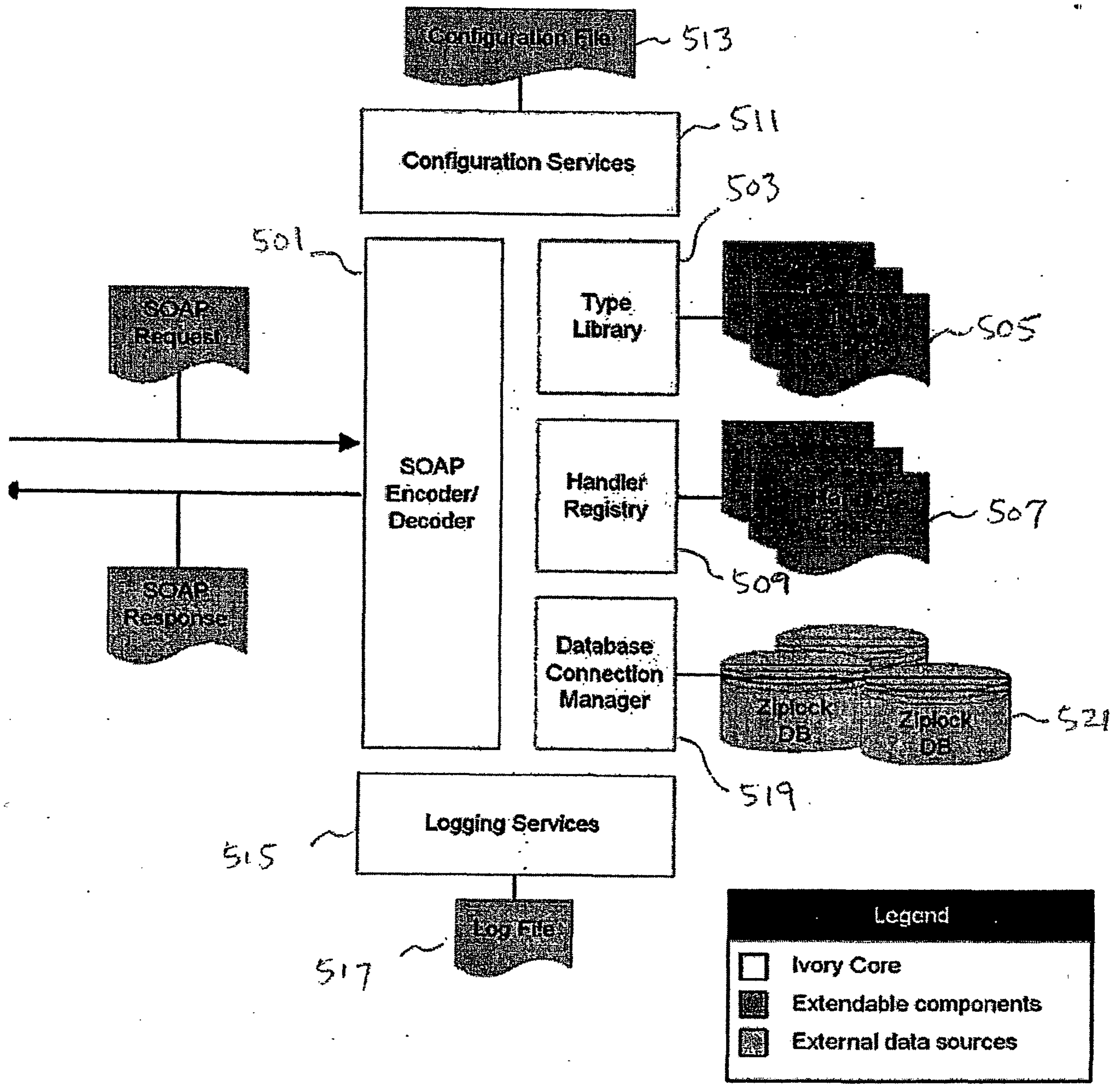

Configuration File
513
Configuration Services
511
503
501
SOAP Request
SOAP Response
SOAP Encoder/ Decoder
Type Library
505
Handler Registry
507
509
Database Connection Manager
Ziplock DB
Ziplock DB
521
519
Logging Services
515
Log File
517
Legend
Ivory Core
Extendable components
External data sources

FIG 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/19007

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G06F 9/54
US CL : 709/330
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 709/330

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5,606,493 A (DUSCHER et al) 25 February 1997 (25.02.1997), see Summary of the Invention.	1-7
Y	US 5,566,302 A (KHALIDI et al) 15 October 1996 (15.10.1996), see Summary of the Invention.	1-7

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:	
"A" document defining the general state of the art which is not considered to be of particular relevance	"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"E" earlier application or patent published on or after the international filing date	"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O" document referring to an oral disclosure, use, exhibition or other means	
"P" document published prior to the international filing date but later than the priority date claimed	"&" document member of the same patent family

Date of the actual completion of the international search 23 October 2001 (23.10.2001)	Date of mailing of the international search report 30 NOV 2001
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703)305-3230	Authorized officer ST. JOHN COURTENAY [illegible] Telephone No. 703 305-3665

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

CORRECTED VERSION

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
3 January 2002 (03.01.2002)

PCT

(10) International Publication Number
WO 02/001356 A1

(51) International Patent Classification⁷: G06F 9/54

(21) International Application Number: PCT/US01/19007

(22) International Filing Date: 12 June 2001 (12.06.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/599,510 23 June 2000 (23.06.2000) US

(71) Applicant: PREVIEW SYSTEMS,INC. [US/US]; 1195 W. Fremont Avenue, Suite 2001, Sunnyvale, CA 94087 (US).

(72) Inventors: TAYLOR, Christopher, S.; 10317 Mary Avenue, Cupertino, CA 95014 (US). KIMMET, Timothy, Gordon; 15660 El Gato Lane, Los Gatos, CA 95032 (US).

(74) Agent: KREBS, Robert, E.; Burns, Doane, Swecker & Mathis, LLP, P.O. Box 1404, Alexandria, VA 22313-1404 (US).

(81) Designated State *(national)*: JP.

(84) Designated States *(regional)*: European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR).

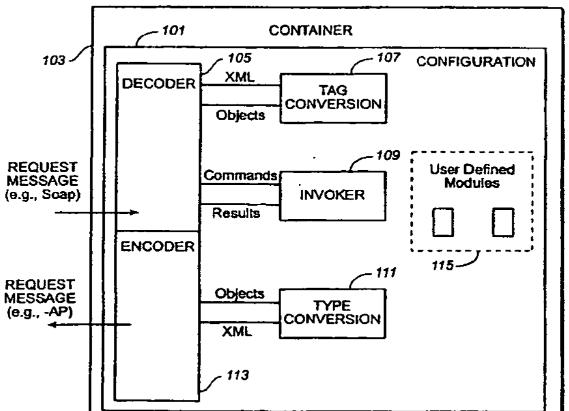
Published:
— *with international search report*

(48) Date of publication of this corrected version:
27 December 2002

(15) Information about Correction:
see PCT Gazette No. 52/2002 of 27 December 2002, Section II

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: TYPE CONVERSION TECHNIQUE FACILITATING REMOTE SERVICE INVOCATION



(57) Abstract: The present invention, generally speaking, provides a messaging framework (101) including a flexible type conversion facility (107, 111) for use in remote method invocation via the Internet. When the framework (101) receives a message, a request message decoder (105) decodes the message; a tag conversion routine (107) converts any parameters into objects, locates the appropriate message handler; and an invoker module (107) invokes the requested method on that handler, and communicates a return value (or exception) via a response message. In the case of the SOAP protocol, for example, the return value (or exception) is encoded into XML and wrapped in a SOAP protocol response message. Dynamic type conversion (111) of parameters into objects is performed using a tag library system (107). In the tag library system (107), special user-defined modules (115) (called "type factories") are handed XML elements for conversion into objects. Unlike a strictly type-conversion architecture such as classic object serialization, there are no dependencies between the XML input and the objects produced by the type factory. Furthermore, the factory is not required to extract all of the content during a conversion, but can pass the raw XML content to the constructed object for later use.

WO 02/001356 A1

TYPE CONVERSION TECHNIQUE FACILITATING REMOTE SERVICE INVOCATION

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to distributed computing, in particular client/server communication and server/server communication.

2. State of the Art

Various techniques enabling software in one location to invoke the services of software at another location are known, including, for example, RPCs (Remote Procedure Calls), Java RMI (Remote Method Invocation), CORBA (Common Object Request Broker Architecture), and Microsoft COM (Common Object Model). Many of these techniques entail substantial complexity. Furthermore, in the case of COM, for example, COM components can only run on Microsoft Operating Systems, and COM requests cannot pass through a firewall. In contrast to the complexity of the foregoing solutions, XML (eXtensible Markup Language) has surged in popularity, due at least in part to its simplicity and flexibility.

The SOAP (Simple Object Access Protocol) protocol is XML-based and enables a client's request to be communicated over HTTP, across the Internet and through firewalls. XML documents are used for encoding of the invocation requests and responses. In the SOAP protocol, a method request is represented as an XML element with additional elements for parameters. The SOAP protocol considerably simplifies remote method invocation. Nevertheless, SOAP makes no provision for type conversion, i.e., mapping between XML and data types usable by the method being invoked and, in the reverse direction, between data types and XML. Two techniques are known. One technique encodes data types through "introspection," i.e., programmatically inspecting an object, walking through its member variables, and turning each into XML. In another technique, no mapping between data types and XML is performed at all. Rather, raw XML is simply passed to the method. Doing so makes for messy code within the method and has a tendency to defeat the discipline of good object-oriented coding practices. Hence, these techniques are fairly cumbersome.

Also, in these prior art mechanisms, versioning issues arise in which, if changes in services are made on the server side, corresponding changes must be made to each individual client to maintain operability. Programmers making changes to services must therefore concern themselves with client/server business logic, potentially a substantial burden.

Therefore, there remains a need for a flexible type conversion facility for use in remote method invocation, e.g., in conjunction with the SOAP protocol.

SUMMARY OF THE INVENTION

The present invention, generally speaking, provides a messaging framework including a flexible type conversion facility for use in remote method invocation via the Internet. When the framework receives a message, it decodes the message, converts any parameters into objects, locates the appropriate message handler, invokes the requested method on that handler, and communicates a return value (or exception) via a response message. In the case of the SOAP protocol, for example, the return value (or exception) is encoded into XML and wrapped in a SOAP protocol response message. Dynamic type conversion of parameters into objects is performed using a tag library system. In the tag library system, special user-defined modules (called "type factories") are handed XML elements for conversion into objects. Unlike a strict type-conversion architecture such as classic object serialization, there are no dependencies between the XML input and the objects produced by the type factory. Furthermore, the factory is not required to extract all of the content during a conversion, but can pass the raw XML content to the constructed object for later use.

BRIEF DESCRIPTION OF THE DRAWING

The present invention may be further understood from the following description in conjunction with the appended drawing. In the drawing:

Figure 1 is a block diagram of one computing environment using the present messaging framework;

Figure 2 is a diagram showing the organization of a tag library used to perform conversion between XML and code objects;

Figure 3 is a conceptual diagram illustrating conversion from XML;

Figure 4 is a block diagram of another computing environment using the present messaging framework; and

Figure 5 is a block diagram showing in greater detail various components of the present messaging framework.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring now to Figure 1, a block diagram is shown of one computing environment using the present messaging framework, referred to as the "Ivory" messaging framework. In Figure 1, the messaging framework (configuration 101) is shown as residing within a container 103. An example of one such container is a "servlet," i.e., an extension similar to a Java applet except that it runs on the server instead of the client. Servlets are described in greater detail in Taylor and Kimmet, *Core Java Web Server*, Prentice Hall, 1999, incorporated herein by reference.

The messaging framework consists of a collection of components which, when initialized to an operational state, form a configuration of the messaging framework. These components include, as shown in Figure 1, a request message decoder 105, tag conversion routines 107, an invoker module 109, type conversion routines 111, and a response message encoder 113. The messaging framework interacts with various user-define modules 115 that perform the desired services.

The message request decoder 105 receives an XML request message, e.g., a SOAP message, decodes the message, and for different tags of the XML message, passes the XML for the tag to a tag conversion routine 107. Tag conversion routines 107 receive as input the XML for a tag and produce as output corresponding code objects (e.g., Java objects). The message request decoder 105 makes these objects available to the invoker module 109 along with a corresponding command. The invoker module 109 receives the command and invokes a corresponding user defined module 115 to perform the actual requested work. The invoker modules 109 receives command results, in the form of objects, from the user defined module 115 and returns these results to the response message encoder 105.

The response message encoder 113 receives the result objects and for different ones of the objects, passes the object to a corresponding type conversion routine 111. Type conversion routines 111 perform the opposite function of tag conversion routines, i.e., receive as input an object and produce as output corresponding XML.

Finally, the response message encoder 113 uses the XML to create an XML message, e.g, a SOAP message.

In general, when a SOAP message is handed to the messaging framework, the framework performs the following steps:

1. Decode the message.
2. Convert any parameters to objects.
3. Locate the appropriate handler.
4. Invoke the method on that handler.
5. Encode the return value (or exception) into XML.

6. Wrap the return value in a SOAP response message.

Whereas SOAP is only a protocol-layer construction, the present messaging framework provides for the separation and distinction of different aspects of remote method invocation, potentially allowing for these different aspects to be addressed by different programmers or developers. For example, an invoker developer would be concerned with what gets done in a request, a tag conversion developer would be concerned with what kinds of data can be passed in a request, and a protocol handler developer would be concerned with how messages get into and out of the messaging framework.

An important feature of the messaging framework is a "Tag Library" system in which special user defined modules (referred to sometimes hereafter as "type factories") are handed XML elements for conversion into objects. Unlike a strict type-conversion architecture like classical object serialization, there are no dependencies between what XML enters the framework and the objects produced by each user defined module. Furthermore, the user defined modules are not required to extract all of the content during a conversion, but can pass raw XML content to the constructed object for later use.

To use the messaging framework, a configuration is first built. In an exemplary embodiment, the configuration process begins by reading an ivory.config file, the content of which is specified in XML format. The name of an XML top-level tag and its associated tag (*toXML*) converter are specified, as well as the name of a Java object and its type (*fromXML*) converter. In the example of Figure 2, two converters are specified: string, which is a primitive type converter for a Java string, and Product, which is a user-defined type for a more complex object.

The messaging framework loads the ivory.config file and creates a tag library (converter library) having two types of converters: *toXML* converters, which map an object type to its corresponding converter, or "type factory," and *fromXML* converters, which map an XML tag and associated elements to its corresponding converter.

In operation, the messaging framework converts objects from XML by matching the top-level XML tag with its associated converter. Referring to Figure 3, an example of shown of how, using the tag library of Figure 2, a top-level Product element is converted from XML into a PreviewProduct using the PreviewProductType converter. The messaging framework performs the following tasks in order to convert the XML representation of the Product into a PreviewProduct Java object:

1. Get the name of the top-level tag, which is "Product."
2. Use the fromXML portion of the tag library to look up the converter for the XML tag with the name "Product." This step renders an instance of the PreviewProductType converter.
3. Invoke fromXML on the converter, which returns a Java object that is an instance of PreviewProduct.

Note that a complex type might contain simple types. In this instance, a converter for the complex type might use a converter for the simple type to help perform the conversion. Note also that converter need not have knowledge of every element to perform a conversion; unknown elements may simply be passed along unconverted to a user defined module for the user defined module to handle. Considerable flexibility therefore results.

Appendix A shows a type converter that extracts text content and returns a String object.

The Ivory framework is not container dependent. Hence, although the HTTP represents a convenient container with which the messaging framework may be used, any other desired system may be used. For example, Figure 4 shows the messaging framework used with email, in accordance with any of various email protocols.

Referring to Figure 5, a block diagram shows in greater detail various components of the present messaging framework. The encoder/decoder 501 performs the functions of the encoder and decoder in the representation of Figure 1. Where the messaging framework is used with the SOAP protocol, for example, the encoder/decoder may represent the SOAP protocol encoding/decoding layer. SOAP messages, as remote method invocations, contain typed data. These data types are converted into objects before leaving the encoding/decoding layer 501.

The responsibility for converting data types into objects falls on the type library 503. The type library 503 performs the functions of tag conversion and type conversion in Figure 1. The type library maps incoming XML data elements to type factories 505. When an XML element arrives that has an associated type factory, that factory builds a Java object from the XML. The type factories are the converter routines of Figure 2.

The purpose of the messaging framework is to make remote method invocation easy to accomplish. In the case of the present messaging framework, remote method invocation is used to invoke the services of request handlers. A handler request is a remote object that exposes specific methods to the messaging framework. Maintaining these objects and routing incoming method calls is the responsibility of the handler registry 509. The handler registry performs the invoker functions of Figure 1. The handler objects correspond to the user defined modules of Figure 1. In an exemplary embodiment, the messaging framework does not require its handler objects to implement a fixed interface. Rather, all a handler object needs to expose a method to the messaging framework is to prefix the word "soap" to the front of any public method. During initialization of the messaging framework, handler objects are loaded and bound into an invoker object. During the binding process, the handler object undergoes introspection, in which all of the public methods of the handler object are inspected by name. Any methods starting with the keyword "soap" are exported for remote invocation purposes as illustrated in the code example of Appendix B.

The configuration services block 511 receives input from a configuration file as described in connection with Figure 2. Rather than using a simple text file for configuring the messaging framework, however, all framework objects are extracted from a configurator. In an exemplary embodiment, the configurator parses a DOM tree for configuration information.

In addition, there is shown in Figure 5 an optional logging services block 515 that produces a log file 517, and an optional database connection manager 519. The database connection manager allows handler objects to make database connections. In the example of Figure 5, the database connection manager connects to a Ziplock database 521 associated with a ZiplockTM electronic software distribution server of the present assignee.

A code example illustrating invocation of the desired method is provided as Appendix C.

It will be appreciated by those of ordinary skill in the art that the invention can be embodied in other specific forms without departing from the spirit or essential character thereof. The presently disclosed embodiments are therefore considered in all respects to be illustrative and not restrictive. The scope of the invention is indicated by the appended claims rather than the foregoing description, and all changes which come within the meaning and range of equivalents thereof are intended to be embraced therein.

What is claimed is:

1. A method whereby one computer remotely invokes a service by another computer, wherein a message is used to construct an object to be used by the other computer in providing the service, the method comprising the steps of:

providing, accessible to the other computer, a tag library comprising, for each of multiple tag values, a named routine for constructing respective corresponding objects;

the one computer producing a message including tags denoting types, including at least one tag denoting one type to be used to construct an object, and, corresponding to each tag, text representing one or more element values, the message including an identifier of a requested service;

decoding the message;

for said one tag, calling a corresponding routine from the tag library, the routine constructing an object using one or more elements values of said one type; and

making the object available to a handler for performing the requested service, and calling the handler.

2. The method of Claim 1, wherein the tag library comprises, for each of multiple data types, a named routine for producing a tag/elements representation of the data type, the method further comprising:

the handler performing the requested service and returning result or status information including at least one data type;

for said one data type, call a corresponding routine from the tag library, the routine producing a representation of the data type as a tag and, corresponding to the tag, text representing one or more element values; and

sending a response message including the tag and element values.

3. The method of Claim 1, wherein the message is a SOAP message.

4. The method of Claim 2, wherein the response message is a SOAP message.

5. The method of Claim 1, wherein the message is an email message.

6. The method of Claim 2, wherein the response message is an email message.

7. The method of Claim 1, wherein, in addition to making the object available to a handler, other tag and element values not used to create the object are made available to the handler.

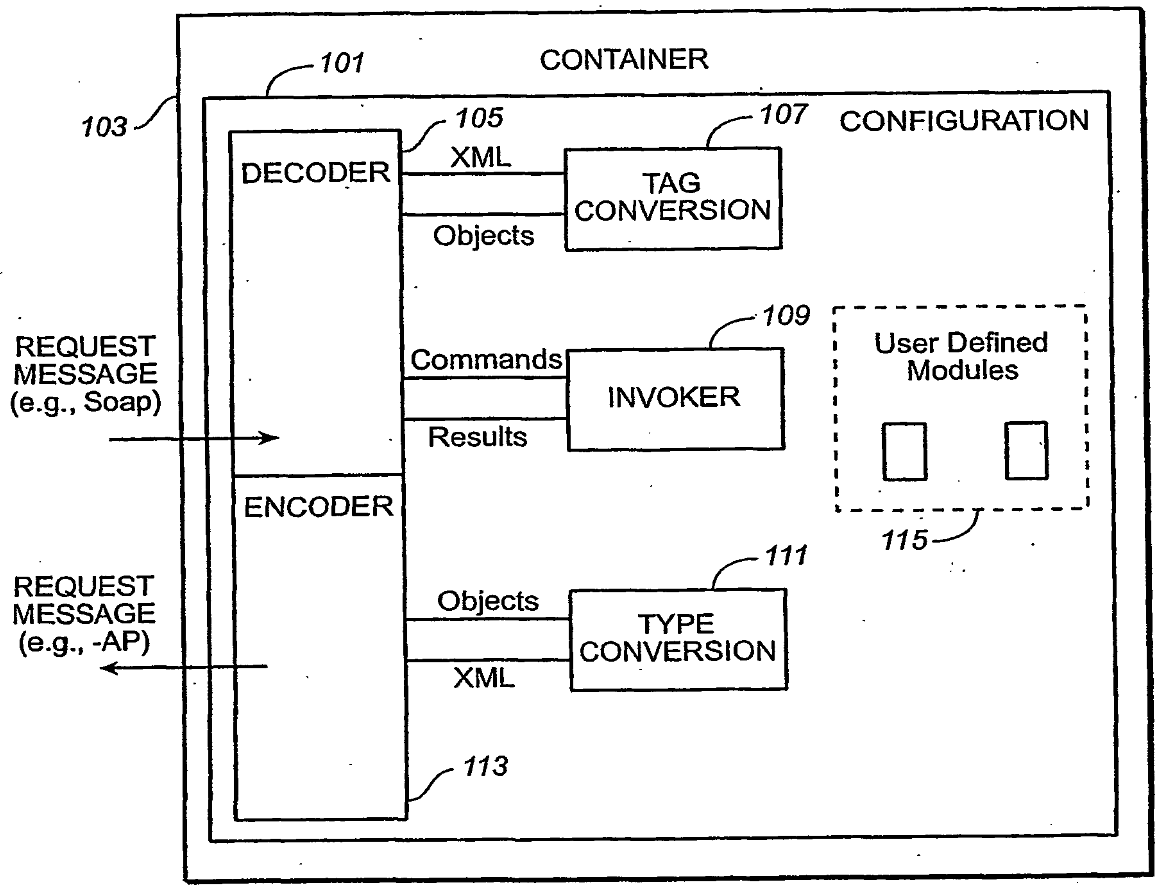
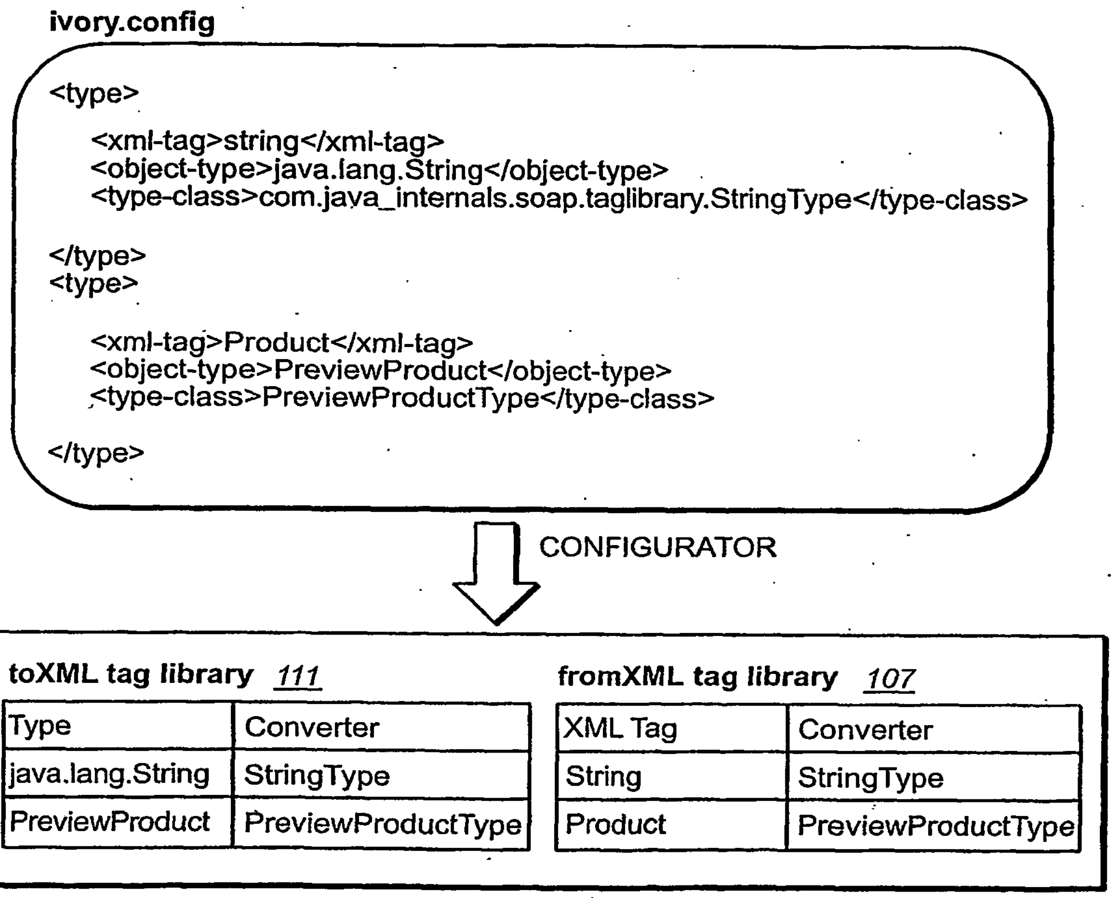

CONTAINER
101
103
CONFIGURATION
105
107
DECODER
XML
TAG CONVERSION
Objects
109
User Defined Modules
Commands
INVOKER
Results
REQUEST MESSAGE (e.g., Soap)
ENCODER
115
111
Objects
TYPE CONVERSION
XML
REQUEST MESSAGE (e.g., -AP)
113

FIG._1



ivory.config

```
<type>

    <xml-tag>string</xml-tag>
    <object-type>java.lang.String</object-type>
    <type-class>com.java_internals.soap.taglibrary.StringType</type-class>

</type>
<type>

    <xml-tag>Product</xml-tag>
    <object-type>PreviewProduct</object-type>
    <type-class>PreviewProductType</type-class>

</type>
```

toXML tag library 111

Type	Converter
java.lang.String	StringType
PreviewProduct	PreviewProductType

fromXML tag library 107

XML Tag	Converter
String	StringType
Product	PreviewProductType

FIG._2

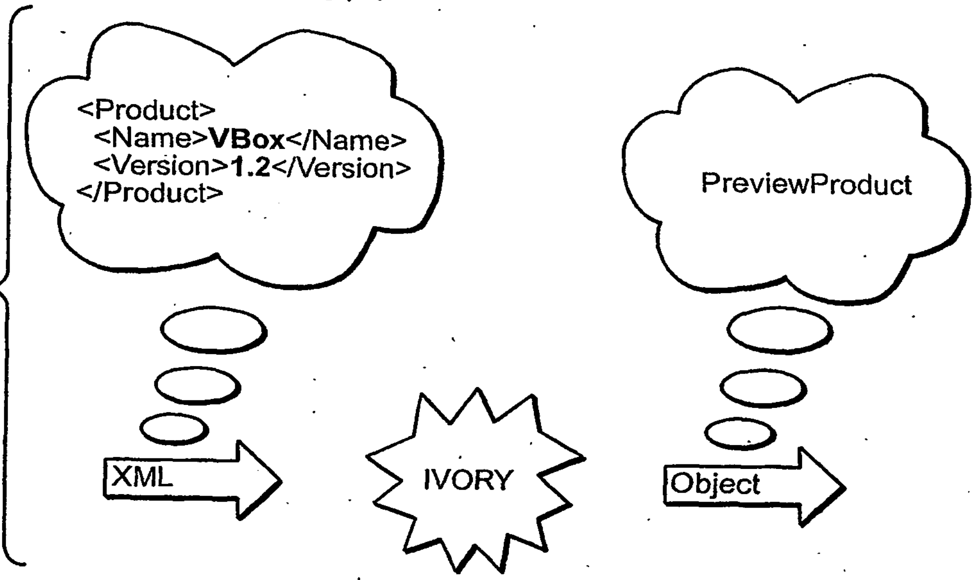

<Product>
<Name>VBox</Name>
<Version>1.2</Version>
</Product>
PreviewProduct
XML
IVORY
Object

FIG._3

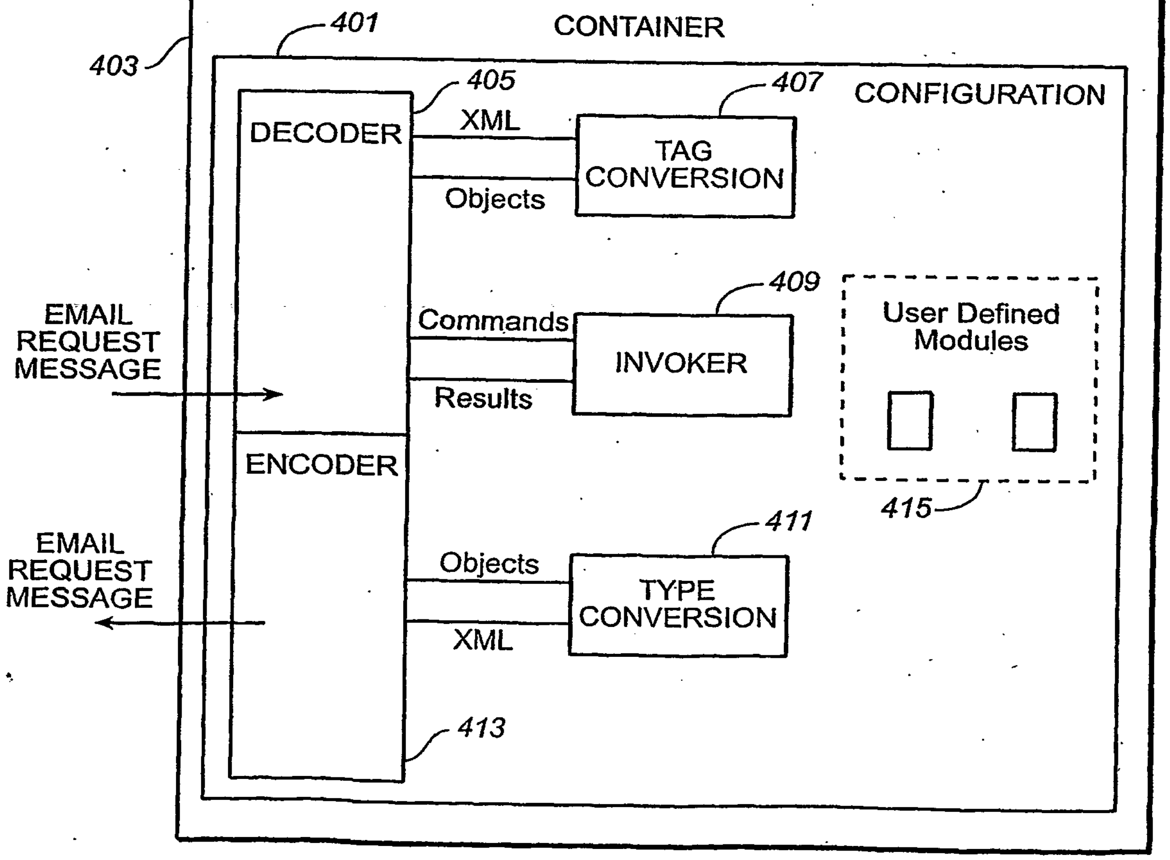

401
CONTAINER
403
405
407
CONFIGURATION
DECODER
XML
TAG CONVERSION
Objects
409
User Defined Modules
EMAIL REQUEST MESSAGE
Commands
INVOKER
Results
ENCODER
415
411
EMAIL REQUEST MESSAGE
Objects
TYPE CONVERSION
XML
413

FIG._4

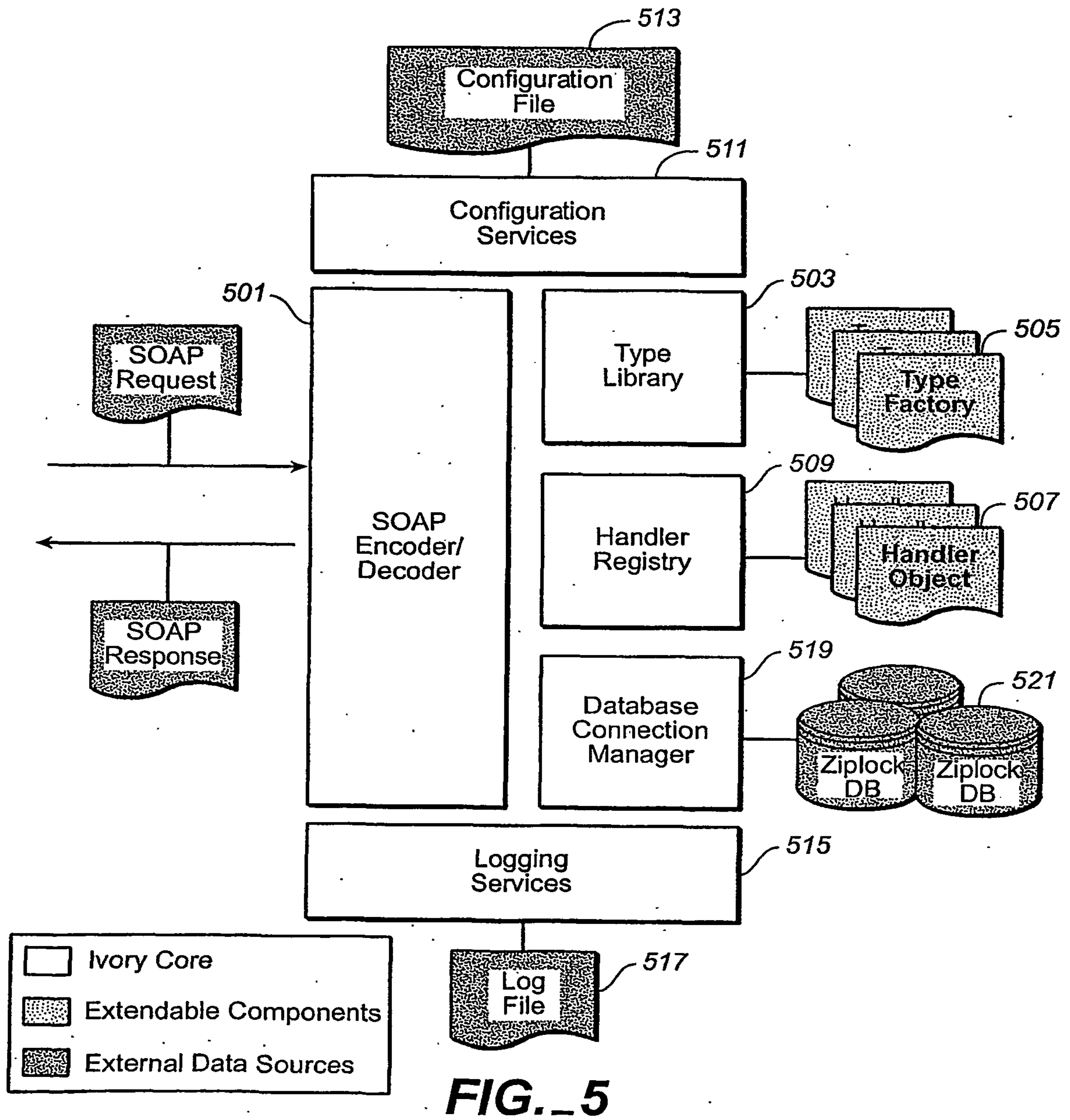

513
Configuration File
511
Configuration Services
501
SOAP Request
SOAP Encoder/ Decoder
SOAP Response
503
Type Library
505
Type Factory
509
Handler Registry
507
Handler Object
519
Database Connection Manager
521
Ziplock DB
Ziplock DB
515
Logging Services
517
Log File
Ivory Core
Extendable Components
External Data Sources

FIG._5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/19007

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : G06F 9/54
US CL : 709/330
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 709/330

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5,606,493 A (DUSCHER et al) 25 February 1997 (25.02.1997), see Summary of the Invention.	1-7
Y	US 5,5[illegible],302 A (KHALIDI et al) 15 October 1996 (15.10.1996), see Summary of the Invention.	1-7

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A"	document defining the general state of the art which is not considered to be of particular relevance
"E"	earlier application or patent published on or after the international filing date
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O"	document referring to an oral disclosure, use, exhibition or other means
"P"	document published prior to the international filing date but later than the priority date claimed
"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&"	document member of the same patent family

Date of the actual completion of the international search
23 October 2001 (23.10.2001)

Date of mailing of the international search report
30 NOV 2001

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703)305-3230

Authorized officer
ST. JOHN COURTENAY James R. Matthews
Telephone No. 703 305-3665